Base32 Encoding and Decoding With iRules

George Watkins, 2011-08-12

Introduction

Anyone that's done any amount of programming has probably encountered Base64 encoded data. Data that is encoded with Base64 has the advantage of being composed of 64 ASCII characters, which makes it portable and readable with virtually any RFC-compliant decoder. Before Base64 became the de facto encoding standard for content, Base32 was the preferred method.

Base32 offers three distinct advantages over Base64: it is case-insensitive, commonly confused characters have been removed (0, 1, and 8 are not included in the Base32 alphabet because they can be mistranscribed as 'O', 'I', and 'B' respectively), and lastly all the characters can be included in a URL without encoding any of them. Here is the full Base32 alphabet:

The RFC 4648 Base 32 alphabet*

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y		
7	H	16	Q	25	Z		
8	I	17	R	26	2	pad	=

*source Wikipedia's Base32 article

The ability to be able to write down a string of encoded data without worrying about case-sensitivity or mistranscription are advantages, but there are some downsides to Base32 encoding. A few of these reasons include: Base32 is approximately 20% less efficient than Base64 on average and it does not handle UTF-16 (full Unicode) elegantly. With those caveats in place, there is still plenty of code today that uses Base32.

Base32 Encoding

Base32 encoding works by processing 40-bit chunks, known as quantums, which are composed of five pieces of 8-bit text (UTF-8). During the encoding process, each 8-bit piece of text is read and converted to its UTF-8 integer value. Integer values can range from 0x00 to 0xFF (8-bit) and are concatenated into a string of bits, then read in 5-bit chunks to obtain the corresponding Base32 alphabet symbol. This action is repeated until the final chunk is processed. If the final chunk is exactly 5 bits, it is processed as is, if it is less than 5 bits, zeros are padded (on the right) to bring the chunk to 5-bits, then it is processed. If the final quantum is not an integral multiple of 40, the final quantum is padded with '=' so that the encoded string has a number of characters which are an integral multiple of 8.

```
array set b32_alphabet_inv {
    0 A  1 B  2 C  3 D
    4 E  5 F  6 G  7 H
    8 I  9 J 10 K 11 L
   12 M 13 N 14 O 15 P
   16 Q 17 R 18 S 19 T
   20 U 21 V 22 W 23 X
   24 Y 25 Z 26 2 27 3
   28 4 29 5 30 6 31 7
}

set input "Hello World!"
set output ""
set l [string length $input]
set n 0
set j 0

# encode loop is outlined in RFC 4648 (http://tools.ietf.org/html/rfc4648#page-8)
```

```
for { set i 0 } { $i < $l } { incr i } {
  set n [expr $n << 8]
  set n [expr $n + [scan [string index $input $i] %c]]
  set j [incr j 8]

  while { $j >= 5 } {
    set j [incr j -5]
    append output $b32_alphabet_inv([expr ($n & (0x1F << $j)) >> $j])
  }
}

# pad final input group with zeros to form an integral number of 5-bit groups, then encode
if { $j > 0 } { append output $b32_alphabet_inv([expr $n << (5 - $j) & 0x1F]) }

# if the final quantum is not an integral multiple of 40, append "=" padding
set pad [expr 8 - [string length $output] % 8]
if { ($pad > 0) && ($pad < 8) } { append output [string repeat = $pad] }

# $output = JBSWY3DPEBLW64TMMQQQ====
puts $output
```

Base32 Decoding

Base32 decoding works in a similar fashion to encoding, just in reverse. By eliminating the need for padding the final bit or quantum, the code is simpler and cleaner. The decoding process begins by removing any '=' padding from the final quantum. Next, the string is looped through character by character extracting the Base32 integer value from each 5-bit character and concatenating the bits into a string. After 8 bits or more bits have been collected, the chunks are then processed in 8-bit chunks converting their 8-bit integer value to its UTF-8 value (full Unicode support is unavailable as the chunk is 8-bit vs. the 16-bits required for Unicode). This processing continues until the Base32-encoded string has been fully read and converted to its respective UTF-8 characters.

```
array set b32_alphabet {
  A 0   B 1  C 2   D 3
  E 4   F 5  G 6   H 7
  I 8   J 9  K 10  L 11
  M 12  N 13 O 14  P 15
  Q 16  R 17 S 18  T 19
  U 20  V 21 W 22  X 23
  Y 24  Z 25 2 26  3 27
  4 28  5 29 6 30  7 31
}

set input JBSWY3DPEBLW64TMMQQQ====

set input [string toupper $input]
set input [string trim $input =]
set l [string length $input]
set output ""
set n 0
set j 0

# decode loop is outlined in RFC 4648 (http://tools.ietf.org/html/rfc4648#page-8)
for { set i 0 } { $i < $l } { incr i } {
  set n [expr $n << 5]
  set n [expr $n + $b32_alphabet([string index $input $i])]
  set j [incr j 5]

  if { $j >= 8 } {
    set j [incr j -8]
    append output [format %c [expr ($n & (0xFF << $j)) >> $j]]
  }
}

# $output = "Hello World!"
puts $output
```

Base32 Encoding and Decoding iRule

If the encoding and decoding functions are combined into a single iRule, you'll wind up with something similar to the code below. This code just logs the encoding an decoding process and is of little use until combined with some input or output data, but here it is:

```
when RULE_INIT {
  set b32_tests {
    JBSWY3DPEBLW64TMMQQQ====
    KRUGS4ZANFZSAYLON52GQZLSEBZXI4TJNZTSC===
    KRUGS4ZANFZSAYJAON2HE2LOM4QHO2LUNAQGCIDQMFSCA33GEA2A====
    KRUGS4ZANFZSAYJAON2HE2LOM4QHO2LUNBXXK5BAMEQGM2LOMFWCA4LVMFXHI5LN
  }

  array set b32_alphabet {
    A 0  B 1  C 2  D 3
```

```
    E 4  F 5  G 6  H 7
    I 8  J 9  K 10 L 11
    M 12 N 13 O 14 P 15
    Q 16 R 17 S 18 T 19
    U 20 V 21 W 22 X 23
    Y 24 Z 25 2 26 3 27
    4 28 5 29 6 30 7 31
  }

  foreach input $b32_tests {
    log local0. "       input = $input"

    set input [string toupper $input]
    set input [string trim $input =]
    set l [string length $input]
    set output ""
    set n 0
    set j 0

    # decode loop is outlined in RFC 4648 (http://tools.ietf.org/html/rfc4648#page-8)
    for { set i 0 } { $i < $l } { incr i } {
      set n [expr $n << 5]
      set n [expr $n + $b32_alphabet([string index $input $i])]
      set j [incr j 5]

      if { $j >= 8 } {
        set j [incr j -8]
        append output [format %c [expr ($n & (0xFF << $j)) >> $j]]
      }
    }

    log local0. "output/input = $output"

    # flip b32_alphabet so that base32 characters can be indexed by a 8-bit integer
    foreach { key value } [array get b32_alphabet] {
      array set b32_alphabet_inv "$value $key"
    }

    set input $output
    set output ""
    set l [string length $input]
    set n 0
    set j 0

    # encode loop is outlined in RFC 4648 (http://tools.ietf.org/html/rfc4648#page-8)
    for { set i 0 } { $i < $l } { incr i } {
      set n [expr $n << 8]
      set n [expr $n + [scan [string index $input $i] %c]]
      set j [incr j 8]

      while { $j >= 5 } {
        set j [incr j -5]
        append output $b32_alphabet_inv([expr ($n & (0x1F << $j)) >> $j])
      }
    }

    # pad final input group with zeros to form an integral number of 5-bit groups, then encode
    if { $j > 0 } { append output $b32_alphabet_inv([expr $n << (5 - $j) & 0x1F]) }

    # if the final quantum is not an integral multiple of 40, append "=" padding
    set pad [expr 8 - [string length $output] % 8]
    if { ($pad > 0) && ($pad < 8) } { append output [string repeat = $pad] }

    log local0. "      output = $output"
    log local0. [string repeat - 20]
  }
}
```

Attach the iRule to a virtual server (preferably not in production) and send some traffic to the virtual. Then check '/var/log/ltm' for the encoded and decoded strings. Ours looked like this:

```
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :        input = JBSWY3DPEBLW64TMMQQQ====
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : output/input = Hello World!
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :       output = JBSWY3DPEBLW64TMMQQQ====
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : --------------------
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :        input = KRUGS4ZANFZSAYLON52GQZLSEBZ
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : output/input = This is another string!
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :       output = KRUGS4ZANFZSAYLON52GQZLSEBZ
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : --------------------
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :        input = KRUGS4ZANFZSAYJAON2HE2LOM4Q
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : output/input = This is a string with a pad
```

```
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : output/input = This is a string with a pad
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :       output = KRUGS4ZANFZSAYJAON2HE2LOM4Q
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : --------------------
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :        input = KRUGS4ZANFZSAYJAON2HE2LOM4Q
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : output/input = This is a string without a
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 :       output = KRUGS4ZANFZSAYJAON2HE2LOM4Q
Dec  8 15:30:43 tmm info tmm[23157]: Rule /Common/base32 : --------------------
```

CodeShare: Base32 Encoder/Decoder iRule

Unicode (UTF-16) Handling

This code sample will not process UTF-16 character sets correctly as they rely on 16-bit integers for their corresponding characters. Reading 16-bit chunks is not supported in RFC 4648 as Unicode was not supported on any of the operating systems that originally used Base32 encoding. Some high-level languages have context aware handling for other character sets, but it is more commonly the exception, not the rule. If you must encode Unicode, you can encode with Base64 first, then encode the output with Base32. Decode in the reverse order.

Conclusion

You're probably asking yourself why in the world anyone would ever want to use Base32 encoding. It was used historically in operating systems that did not support case-sensitivity, but those days are long gone. Aside from the reasons listed in the introduction, the main advantage and the reason it is still used today is that it can be transcribed via the phonetic alphabet with less effort and fewer mistakes than Base64. Symmetric keys are often exchanged using this method, which is the use-case that we ran into and spawned this Tech Tip. We don't want to give away too much, but there is another Tech Tip around the corner that specifically uses this encoding. Until then, happy (en)coding...

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com